Touch Based POMDP Manipulation via Sequential Submodular Optimization

Ngo Anh Vien and Marc Toussaint

***Abstract*— Exploiting the submodularity of entropy-related objectives has recently led to a series of successes in machine learning and sequential decision making. Its generalized framework, adaptive submodularity, has later been introduced to deal with uncertainty and partially observability, achieving near-optimal performance with simple greedy policies. As a consequence, adaptive submodularity is in principle a promising candidate for efficient touch-based localization in robotics. However, applying that method directly on the motion level shows poor scaling with the dimensionality of the system. Being motivated by hierarchical partially observable Markov decision process (POMDP) planning, we integrate an action hierarchy into the existing adaptive submodularity framework. The proposed algorithm is expected to effectively generate uncertainty-reducing actions with the help from an action hierarchy. Experimental results on both, a simulated robot and a Willow Garage PR2 platform, demonstrate the efficiency of our algorithm.**

I. Introduction

Efficient object manipulation typically requires a plan of actively contact-seeking actions to reduce uncertainty over the true environmental model, e.g., poses and positions of objects to be grasped or touched as well as obstacles. While vision is usually the primary sensor information to reduce uncertainty, in this paper we focus on haptic feedback. Humans are extremely skilled in object manipulation also when deprived of vision. We therefore consider the scenario of a robot entering a dark room and seeking for an object on a table as shown in Fig. 1. The only sensor information are force/torque signals in the end-effector. This task is very challenging not only for robots but also humans, as the task includes a lot of uncertainty [1]. To solve this type of tasks, humans usually seek contacts with objects in order to disambiguate uncertainty. In principle, this task can be mathematically formulated as a partially observable Markov decision process (POMDP) whose state space consists of hidden states that can only be inferred through observations. For instance, the poses and locations of objects are not directly observable, but sensed contact forces are observable [2]–[4]. The resulting POMDP would have high-dimensional continuous state, observation and action spaces and a very long horizon if we consider low-level robot control as the action level. Solving this general POMDP is known to be PSPACE hard [5]. Therefore approximation and heuristic methods are needed.

In this paper, we propose methods to approximate and efficiently solve the problem of manipulation under uncertainty via *tactile feedback*. We approximate the problem by using high-level actions to deal with the long horizon problem [6], [7], i.e. macro actions, and use a sample-based approach to deal with both high-dimensional and long horizon problems [8]. Though being approximated, naively applying standard POMDP solvers will still take relatively high cost to find a good policy. However, by re-designing the objective function to be submodular (i.e. diminishing returns and monotonic) we can efficiently apply the recently introduced adaptive submodularity framework, whose greedy policies are proved to guarantee near-optimal performance [9], [10].

This work was supported by the EU-ICT Project 3rdHand 610878.

Ngo Anh Vien & Marc Toussaint are with the Machine Learning & Robotics Lab, University of Stuttgart, Germany, {vien.ngo;marc.toussaint}@ipvs.uni-stuttgart.de

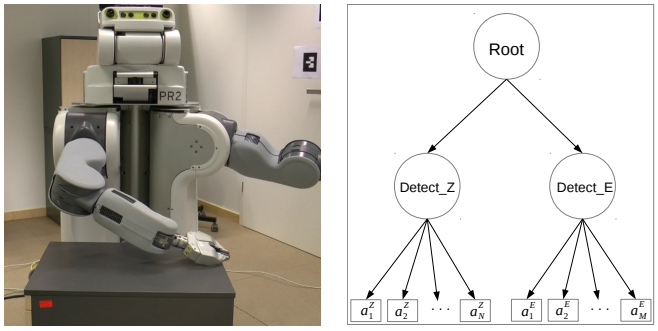


Fig. 1. The peg-in-hole-like task: (left) A robot is localizing a table in order to press a button at the center of the table; (right) An action hierarchy: Detect_Z is a subtask to detect the table's height, Detect_E is a subtask to detect the table's edges.

Both methods, POMDP and adaptive submodularity, can seek contacts with objects for uncertainty disambiguation. We go one step further in combining them: To address more complex task in which contacts are harder to be made we propose to decompose the task into smaller sub-tasks, as in hierarchical POMDP planning [11]. Each sub-task now corresponds to exactly one adaptive submodular optimization task.

In summary, our contributions are three-fold:

- We integrate the benefit of hierarchical POMDP planning, using action decomposition, into the existing adaptive submodularity formulation [10]. The integration is expected to make adaptive submodularity able to tackle more complex tasks in which many actions might not return contact information. Such actions do not help in uncertainty disambiguation, as establishing contacts is the key to success in manipulation tasks. Action decomposition like in hierarchical POMDP is expected to guide the contact-seeking search better.
- The action set can be either seeking contacts as in the previous method [10] or keeping contacts. All those actions are defined similarly to standard macro actions in hierarchical POMDP, which are multiple-step actions, and able to terminate under certain conditions.
- Action selection at the higher level of subtasks can be effectively optimized via POMDP solvers with approximate models of the subtasks [11], or via adaptive submodularity with suitable cost functions [10].

In section II, we briefly review background knowledge about POMDP and submodularity. Next, section III describes

how to integrate an action hierarchy of POMDP into an adaptive submodularity framework. Experiment results are described in section IV. Finally, our conclusion with some remarks is given in section V.

II. Background

In this section, we briefly give an introduction to the POMDP framework and adaptive submodular optimization.

A. Partially Observable Markov Decision Process

Robot manipulation under uncertainty problems can in principle be formulated as a partially observable Markov decision process (POMDP). A POMDP is 7-tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{C}, \gamma\}$, where $\mathcal{S}, \mathcal{A}, \mathcal{O}$ are state, control action, and observation spaces. The transition function $\mathcal{T}(s, a, s') = p(s'|s, a)$ defines a probability of next states if taking action a at state s. The observation function $\mathcal{Z}(s', a, o) = p(o|s', a)$ defines a probability of observations. The cost function is $\mathcal{C}(s, a, s')$, and the parameter γ is a discount factor. An agent must find an optimal policy, which is a mapping $\pi : \mathcal{H} \mapsto \mathcal{A}$ from the history space to the action space, that minimizes the cumulative discounted costs

$$\rho(\pi) = \mathbb{E}\{\sum_t \gamma^t c_t\} \tag{1}$$

A history $h_t \in \mathcal{H}$ is a sequence of t pairs of actions and observations $\{a_0, o_0, a_1, o_1, \ldots, a_{t-1}, o_{t-1}\}$.

In our example problem in Fig. 1, the states are $s \in \mathbb{R}^{n+n_e}$, where the robot's joint configuration is $x \in \mathbb{R}^n$ (assumed to be observable), and the environment state is $e \in \mathbb{R}^{n_e}$ that is unobservable to the robot (i.e. the environment model, e.g. table position and size, object location). The control actions a are motor commands computed by the operational space/force controller. Observations o are sensed forces of a F/T sensor at the wrist of the PR2 robot's arm. Alternatively, one can model observations as binary feedback, i.e. contacts. Based only on a sequence of contacts, the robot should be able localize the table to accomplish his task. Assuming the robot arm always starts above the table, an optimal policy might look like this: the robot first goes down from top until sensing contact with the table, at which point the table's surface is localized. His next optimal macro action is moving sideways while still keeping the contact with the table's surface plane, until the contact vanishes. The robot could effectively infer the table's edges at those contact-losing positions. By finding more similar points at edges, the robot could disambiguate uncertainty of the table's size, location, and orientation. However, finding such an approximately optimal policy for a POMDP is a non-trivial task [3], [8], [12], which is further proved to be NP-hard [13].

B. Adaptive Submodularity

In the case of a submodular and monotonic objective function, a greedy strategy can be guaranteed to achieve a near-optimal performance [14], [15]. Consequently, submodular optimization has recently been widely applied in machine learning [16] because of its efficiency and simplicity. Later, submodulatiry was generalized to adaptive planning, hence named adaptive submodularity [9]. In this adaptive setting, the state is unobservable and observations are generated by actions. This framework is a special formulation of a POMDP in that the state is not influenced by actions. In other words, the transition probability is supposed to be $p(s'|s, a) = \delta_s(s')$. Below, we describe this framework in detail.

Assume that the true underlying state is fixed to be s^*; in our example these are the unknown parameters of the table. There is an observation function, also called realization, $\phi : \mathcal{A} \mapsto \mathcal{O}$. For instance, after executing an action we observe contacts with a part of the table. After choosing an action $a \in \mathcal{A}$, an observation $\phi(a)$ is observed. As the realization is initially unknown, we can denote by Φ a random realization. Analogous to maintaining the full history $h \in \mathcal{H}$ in the POMDP case, in adaptive submodularity we maintain a partial realization $\psi \subseteq \mathcal{A} \times \mathcal{O}$ where $(a, o) \in \psi$ if $\phi(a) = o$ has previously been observed. Denote by $\text{dom}(\psi) = \{a : \exists o, (a, o) \in \psi\}$ the domain of ψ. If a realization ϕ and a partial realization ψ are equal in the whole domain of ψ, ψ is said to be consistent with ϕ, hence denoted as $\phi \sim \psi$. If two partial realizations ψ_1 and ψ_2 both are consistent with ϕ, and $dom(\psi_1) \subseteq dom(\psi_2)$, then ψ_1 is said to be a subrealization of ψ_2. The corresponding random variable of a partial realizations is Ψ. Summing up, we can write the posterior over the realization ϕ conditional on a partial realization ψ as $p(\phi|\psi) = p(\Phi = \phi|\Phi \sim \psi)$. This is similar to the belief representation in POMDP.

Further, let us define f as a set-function $f : 2^{\mathcal{A}} \times \mathcal{O}^{\mathcal{A}} \mapsto \mathbb{R}$, mapping from selected actions and observed observations to a real utility value.

1) Submodularity: First, we briefly describe non-adaptive submodularity. A function f is said to be *submodular* if satisfying the condition of diminishing returns. More specifically, if X and Y are two sets with $X \subseteq Y \subseteq \mathcal{A}$, $a \in \mathcal{A} \setminus Y$, the condition of the diminishing returns is

$$f(Y \cup \{a\}) - f(Y) \leq f(X \cup \{a\}) - f(X) \ . \tag{2}$$

This means that adding an item a to a set X gains more or the same amount of value than adding the same item into its superset Y. The function f is *monotonic* if

$$f(X \cup \{a\}) - f(X) \geq 0 \tag{3}$$

The objective is to find an optimal subset $A^* \subseteq \mathcal{A}$ such that

$$A^* \in \arg\max_{A \subseteq \mathcal{A}} f(A, \phi), s.t. |A| \leq k. \tag{4}$$

where k is a budget constant that limits the number of selected actions. Nemhauser et. al. [14] proved an important result of monotonic submodular functions, namely that a greedy policy which simply chooses

$$A_{i+1} = A_i \cup \{\arg \max_{a \in \mathcal{A} \setminus A_i} f(A_i \cup \{a\})\} \tag{5}$$

is guaranteed a near-optimal performance, i.e., $f(A_k) \geq (1 - 1/e) \max_{|A| \leq k} f(A)$, where starting with $A_0 = \emptyset$.

2) Adaptive Submodularity: In the case of uncertain and partial observations, agents are required to make sequential decisions adaptively [9]. The expected gain if adding an action is captured as

$$\Delta(a|\psi) = \mathbb{E}\big[f(A \cup \{a\}, \Phi) - f(A, \Phi)|\Phi \sim \psi\big] \tag{6}$$

This measures the expected divergence if adding one action a into the executed set of actions A, given a partial realization ψ, for a fixed Φ.

The function f is *adaptive submodular* if $X \subseteq Y \subseteq \mathcal{A}$, $A_i \in \mathcal{A} \setminus Y$, and the inequality

$$\Delta(a|Y) \leq \Delta(a|X) \tag{7}$$

holds for any sequence of observations returned after executing a. The *adaptive monotonicity* is similarly defined as adding more action and observation does not make the

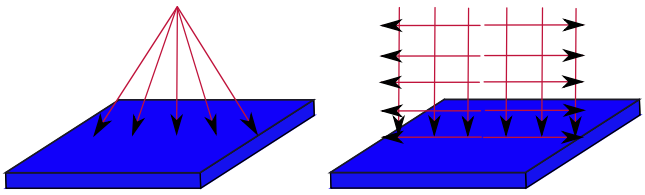


Fig. 2. Action sets of two naive application of adaptive submodularity, assuming the motion is in 2-D y,z-plane).

function value decrease

$$\mathbb{E}\big[f(X,\Phi(X))|\Psi] \leq \mathbb{E}[f(X\cup\{a\},\Phi)|\Psi(X),\Psi(a)=o\big] \tag{8}$$

Golovin and Krause [9] proved a similar result for adaptive monotonic and adaptive submodular functions: The greedy policy π which chooses

$$A_{i+1} = A_i \cup \{\arg\max_{a\in\mathcal{A}\setminus A_i} \Delta(a|A_i)\} \tag{9}$$

is guaranteed a near-optimal performance of $f(\pi) \geq (1-1/e)f(\pi^*)$, where $f(\pi)$ means the value of f after adaptively choosing k actions, and π^* is an optimal policy.

Assuming that the state or realization is not changed by the selected action, one could directly apply adaptive submodularity to solve the problem, as a special form of a POMDP. For instance, Javdani et. al. [10] formulated this problem as one adaptive submodularity optimization problem, then employed the *information gain* metric. They constructed an action set that are optimized trajectories of linear motions composed of sampled starting poses and movement direction vectors. Applying this idea to our problem, one must generate a large number of trajectories that could effectively reach all edges of the table in order to localize the table. Each trajectory corresponds to different linear movement's angles, as shown in Fig. 2 (left). Using such linear motions, the observations of touching a table's edge might be very rare to establish. This would render many optimized trajectories unsuccessful (terminate without making any contacts). Alternatively, the trajectory set might consist of trajectories moving orthogonally downward and trajectories moving horizontally at many different starting point on the z-axis, as depicted in Fig. 2 (right). This alternative would yield the same amount of wasteful optimized trajectories, as the robot does not know which uncertainty, e.g. height or edges, should be disambiguated by which action (trajectory). In the worst case, both mentioned applications of the method in [10] would have to execute all actions in the budget in order to either accomplish the task luckily or fail completely.

Considering how humans might solve the example task in Fig. 1, we should use one high-level strategy that first finds the table's surface, then slides the hand sideways while keeping contacts with the table to localize all edges. This strategy is known as hierarchical POMDP planning [11]. The next section discusses how to integrate the idea of hierarchical POMDP planning into an adaptive submodular framework. The integration will render adaptive submodularity more applicable in manipulation tasks.

III. POMDP Manipulation as Sequential Submodular Optimization

In this paper, we propose a new framework that combines the advantages from hierarchical POMDP planning and adaptive submodular optimization. Similar to hierarchical POMDP planning, an action hierarchy is given by a domain expert as drawn in Fig. 1 (on the right). Each parent node corresponds to one high-level subtask, e.g. $Detect\_Z$ and $Detect\_E$. Each leaf node corresponds to one optimized trajectory like ones in [10]. The trajectory is optimized online when its parent node is selected at a certain starting point. For instance, the example shown in Fig. 1 (left picture) is solved by designing a 3-level action hierarchy (right picture). Consider a policy at the ROOT node that chooses the subtask $Detect\_Z$ to detect the table's height. Solving the subtask $Detect\_Z$ would amount to a smaller localization or motion planning problem that is exactly an instance of adaptive submodular optimization. As in [10], one could construct an action set of optimized trajectories that have different sampled starting poses and one direction vector perpendicular to the table's surface. At the termination of the subtask $Detect\_Z$, the table's height, i.e. table's surface plane, gets localized. If the next subtask is $Detect\_E$, sampling different horizontal movement directions is needed to continue to localize the table's edges. Trajectories as child nodes for $Detect\_E$ are optimized to keep contacts with table's surface until losing contacts. The cost function of the trajectory optimization problem is a weighted sum of obstacle collision, termination, and transitions [17].

A. Problem Formulation

Suppose we are given an action hierarchy $H = \{T_1, \cdots, T_k\}$ consisting of k actions that are either subtasks (compound actions, or options, or macro actions) or primitive trajectories (primitive actions). We make the following assumptions over this action hierarchy.

Assumption 1: Subtasks that consist only of primitive actions as children nodes should be designed such that they satisfy the assumptions of adaptive submodular optimization.

This assumption allows us to use adaptive submodularity at the subtask level right above the primitive action level. As subtasks requires to execute many actions, greedy policies are very efficient.

Assumption 2: Subtasks are defined to disambiguate one or more uncertainty factors.

This assumption implicitly defines termination conditions of the subtasks. According to Assumption 1, we restrict the leaf nodes to be primitive trajectories. Hence, each subtask that consists only of primitive actions (primitive trajectories) correspond to one standard adaptive submodular optimization problem like the action set in [10]. We call them *lower-level* subtasks. Action selection at those subtasks means to simply follow greedy policies.

First, we describe how those subtasks are optimized (similar to the one in [10]), then we describe how to optimize *higher-level* subtasks that contains either *lower-level* subtasks or other *higher-level* subtasks. A subtask might help to localize only a particular part of the total realization space. For instance, $Detect\_Z$ helps localizing the table's height, it always leaves other parts of the realization space intact.

B. Lower-Level Subtasks

Each *lower-level* subtask $\mathcal{T}_i$ corresponds to one adaptive submodular optimization problem. Supposing that the subtask T_i has an action set $\mathcal{A}_i = \{a^i_j\}_{j=1}^{N_i}$. The objective

function for each *lower-level* subtask could be defined as either the averaged cost or the worst case cost as

$$C_{avg}(\pi_i) = \mathbb{E}_{\Phi_i}\Big[c\big(A_i, \Phi\big)\Big] \tag{10}$$

$$C_{wc}(\pi_i) = \max_{\phi} c\big(A_i, \phi\big) \tag{11}$$

where $c(A_i)$ is the incurred cost of executing A_i; $A_i(\pi, \Phi) \subseteq \mathcal{A}_i$ denotes the set of selected actions by policy π_i that is used to localize or to accomplish subtask i. To find an optimal policy π_i for task i, we formulate the following optimization problem

$$\begin{aligned} \text{min.} \quad & C_{avg}(\pi_i) \quad (\text{or } C_{wc}(\pi_i)) \\ s.t. \quad & f(A_i, \phi) \geq q, \quad \forall \phi \end{aligned}$$

The constraint is to achieve at least a value q of the utility function. The utility function in our manipulation task is information gain. The information gain when executing an action a_j^i is

$$\Delta(a_j^i; \Phi) = H(\Phi) - \mathbb{E}_o\big[H(\Phi|o)\big] \tag{12}$$

where o is an observation obtained after executing action a_j^i during subtask i. The observation probability is computed like in [10] that is one normal distribution whose mean is the time of contact with the true model embedded in action j of task i (we construct each action as a trajectory whose terminal pose is based on a particular model sampled from the prior or realization ϕ). Specifically, if the action a_j^i is supposed to have contact at time o_j^i, then the probability of receiving a certain observation o is

$$p(o|\phi) \propto exp\Big(-\frac{|o - o_j^i|}{2\sigma^2}\Big)$$

As the utility function $f(A, \phi) = \Delta(a; \Phi)$ has been proven to be adaptive monotonic and adaptive submodular [9], a greedy policy that chooses actions online using the rule

$$\arg\max_{a \in \mathcal{A}_i} \frac{\Delta(a; \Phi)}{c(a)} \tag{13}$$

is guaranteed to obtain near-optimal performance.

We estimate the model as a multivariate normal distribution over the d-dimensional unknown parameters representing the environment's unknown model. Given a set of particles Φ (each particle represents one model of the world, therefore one trajectory reaching to a particular terminal position can be specified and optimized without uncertainty), the posterior distribution is approximated as $p(\phi|o, \psi) \approx \mathcal{N}(\phi; \mu_o, \Sigma_o)$. Then its differential entropy is computed as

$$H(\Phi|o) \approx \frac{1}{2}\log\big((2\pi e)^d |\Sigma_o|\big)$$

C. Higher-Level Subtasks

As the *higher-level* subtasks might consist of non-primitive actions, the action selection is non-trivial for either POMDP or adaptive submodularity. We now describe two extensions of POMDPs and adaptive submodularity to adapt to the action selection at the *higher-level* subtasks.

1) POMDP: The primitive actions are, in our framework, trajectories and therefore different to standard primitive actions in hierarchical POMDP frameworks. However, we can cast each trajectory as one primitive action with a subsumed cost, then employ the action selection strategy of a POMDP planner. In this section, we propose to use POMDP planning in order to select among high-level actions. In order to do that, we need to approximate models of all higher-level subtasks $\mathcal{T}_i$, i.e. transition, observation probabilities and termination conditions, and execute one-step lookahead to evaluate each child action [11].

According to Assumption 2 in Section III-A each subtask is defined to disambiguate particular uncertainty terms, therefore the termination condition of each *higher-level* subtask is $p(\Phi = \phi_i|\psi) = 1-\epsilon$, where ϵ is a small threshold value; and $p(\Phi \neq \phi_i|\psi) = \epsilon$. This means the realizations are localized at the true partial realization ϕ_i with a high probability. And the task fails to localize the intended uncertainty factor with a small probability ϵ. The parameter ϵ explicitly limits the failure probability of a subtask, therefore it depends on the failure probability of its children actions. In our case, it depends on how we implement the *lower-level* subtasks. Each *lower-level* subtask's failure probability is bounded by the error bound of the greedy policy's performance that has been provided in adaptive submodularity. Therefore, we can recursively compute ϵ for each parent subtask in bottom-up ordering.

Further, as state transitions are static, we can directly estimate the transition function over belief states, i.e. $b_t = p(\phi|\psi)$. Relying on the termination conditions, the belief transition conditioned on the selected subtask T_i and current partial realization ψ is

$$p(b_{t+1}|b_t, T_i) = \begin{cases} 1-\epsilon & \text{, if } b_{t+1} = p(\phi_i|\psi) \\ \epsilon & \text{, if } b_{t+1} = b_t \end{cases}$$

This means that after the task i terminates with a success probability of $1-\epsilon$, the next belief must clearly represent this success with the same certainty probability. Otherwise, the belief transition represents the unsuccessful localization by staying intact at the previous belief. Starting from an initial belief b_0, the belief tree is relatively small. For instance, if a task T_i has N_i children actions and the horizon is set to k, the tree consists of $(2N_i)^k$ nodes. This number is rather small, a standard value iteration for belief MDPs [18] could be used. In the ideal case, if ϵ is set very small when each lower-level subtask is solved using a large amount of samples, we could set k equal to N_i. Because we have assumed that a task terminates when all child actions have been executed or the intended uncertainty factors are localized before resorting to all actions.

One could avoid estimating the models in case of very complex domains with a complex action hierarchy by using a sample-based method for hierarchical POMDPs [7]. Each macro action corresponding to one high-level action $\mathcal{A}_i$ terminates when all actions in the set $\mathcal{A}_i$ are already executed or the terminal conditions as in Assumption 2 are satisfied.

2) Adaptive Submodularity: The action hierarchy is designed specifically to satisfy the assumptions of adaptive submodularity, and each *higher-level* action does not affect the transition of the environment's state. We propose to purely use adaptive submodularity to choose actions at all levels, though the action set at the *higher-level* subtasks is small. Information gain when executing an action a_j^i at subtask T_i is

$$\Delta(a_j^i; \Phi) = H(\Phi) - \mathbb{E}_o\big[H(\Phi|o)\big] \tag{14}$$

We select actions greedily as

$$a = \arg\max_{a' \in \mathcal{A}_i} \frac{\Delta(a'; \Phi)}{\mathbb{E}\big[c(a')\big]} \tag{15}$$

The difference to the standard adaptive submodularity is that we use the expected cost of actions. We take the example described in Section 1 to show how the algorithm works.

First, we describe how we construct the sets of trajectories for each high-level action. Generally, localizing one partic-

ular uncertainty factor requires to sample many different trajectories. In the peg-in-hole-like task, we initially sample a set of 5-tuples $\phi = \{w, l, x, y, z\}$ (particles) that are width, length, positions of the table. There are two *lower-level* subtasks: *Detect_Z* and *Detect_E*, and one *higher-level* subtask: *Root*. For the *Detect_Z* subtask, we construct a set of N_Z trajectories that starts at different starting points from the prior: $s_1 \sim p_1(starting)$. For each pair (s_1, ϕ), we optimize a trajectory to go down straightly from s_1 until sensing the table of height h or the floor. For the *Detect_E* subtask, we construct a set of N_E trajectories that starts at different starting points from the prior: $s_2 \sim p_2(starting)$. For each pair (s_2, ϕ), we optimize a trajectory to go in parallel with the floor from s_2 until observing a change of observations or discovering a side boundary limit. If s_2 starts on the table's surface, the robot should be able to disambiguate both height and edge uncertainties. Therefore, the information gain of choosing *Detect_E* first at *Root* node is higher than that of choosing *Detect_Z* if ignoring the cost $\mathbb{E}\big[c(a')\big]$. However, when taking the cost of actions into account, the cost for each subtask is defined as the average number of taken actions until termination,

$$\mathbb{E}\big[c(\textit{Detect\_E})\big] = \frac{1}{N_E}\sum_{i=1}^{N_E} i = \frac{N_E+1}{2} \tag{16}$$

$$\mathbb{E}\big[c(\textit{Detect\_Z})\big] = 1 \tag{17}$$

and the robot will select *Detect_Z* first. The example is in an ideal situation with an assumption that the robot hand always starts on top of the table.

D. Sequential Adaptive Submodularity Algorithm

The full adaptive submodularity algorithm with action decomposition is pictorially depicted in Algorithm 1. For each subtask a, we denote A_a its executed action set, ψ_a its partial realization. This algorithm uses adaptive submodularity to select actions at all levels. If a is a *higher-level* subtask, one could alternatively replace the action slection strategy at step 16 by the policy computed offline by a POMDP solver as described in Section III-C.1. For a sketch of the algorithm, if an action is an optimized trajectory, it is directly executed then returns an observation as the result. Otherwise, we first construct an action set $\mathcal{A}_a$ for the subtask a as explained in step 5. Steps from 8 to 20 try to solve one particular adaptive submodular problem depending on different types, that would result in a sequence of observations as represented by **obs**.

Algorithm 1 Sequential adaptive submodularity with an action hierarchy

SeqASUB(a)

1: **if** a is a primitive action (an optimized traj.) **then**
2: Execute a, observe an observation $\Phi(a)$
3: Return: $\Phi(a)$
4: **else**
5: Sample starting states and linear motion vectors for task a. Using constrained trajectory optimization to construct the set $\mathcal{A}_a$.
6: $A_a = \emptyset; \psi_a = \emptyset$
7: $\mathbf{obs} = \emptyset$
8: **while** a not terminates **do**
9: **if** a is a *lower-level* subtask **then**
10: Select a_j using Eq. 13
$$a_j = \arg\max_{b \in \mathcal{A}_a / A_a} \frac{\Delta(b;\Phi)}{c(b)}$$
11: Set $o = \mathbf{SeqASUB}(a_j)$
12: Set $A_a = A_a \cup \{a_j\}$
13: Update $\psi_a = \psi_a \cup \{a_j, o\}$
14: $\mathbf{obs} = \mathbf{obs} \cup \{o\}$
15: **else**
16: Select a_j using Eq. 15:
$$a_j = \arg\max_{b \in \mathcal{A}_a / A_a} \frac{\Delta(b;\Phi)}{\mathbb{E}[c(b)]}$$
17: Set $\mathbf{o} = \mathbf{SeqASUB}(a_j)$
18: Set $A_a = A_a \cup \{a_j\}$
19: Update $\psi_a = \psi_a \cup \{a_j, \mathbf{o}\}$
20: $\mathbf{obs} = \mathbf{obs} \cup \{\mathbf{o}\}$
21: **end if**
22: **end while**
23: Return: **obs**
24: **end if**

IV. Experiments

We evaluate the proposed method in two problems on both a simulated 7-DoF KUKA arm (see Fig. 6) and a PR2 physical robot systems. The first problem is simply to localize the table's height in order to find an object hovering 12cm above the table. This task has one uncertainty factor that is the table's height, called *hand-table* problem. This problem has been studied previously in [1] whose method is based on constrained trajectory optimization using uncertainty funneling and contact interaction rewards. The second problem is *peg-in-hole-like* in which the robot is uncertain over the position and size of the table. There is an imagined hole that is situated at the center of the table. Therefore, the robot has to localize the table before it can reach the hole successfully.

In each problem, we sample N particles of Φ from a multivariate normal distribution prior $\mathcal{N}(\mu, \Sigma)$. After each action execution, we compute a posterior as a particle set of Φ, then do resampling in order to maintain a fixed number of particles $|\Phi| = N$. We use constrained trajectory optimization [17] to optimize the trajectory for each sampled model.

A. Experiment Setting

1) Hand-Table: We sample 1000 height hypotheses $\Phi \in \Re$ from a normal distribution $\mathcal{N}(0.65, 0.1)$ (in meters). An action set consists of 10 optimized trajectories that are linear motions made by 10 different starting poses on the z-axis. Those actions can guarantee finding the table's surface. This task consists of only one *Root* node (*Detect_Z*) that is one *lower-level* subtask. Therefore, only *lower-level* subtask is used that corresponds to one adaptive submodular optimization problem. The computational time for each trajectory is approximately 0.1 seconds in a personal computer.

We compare the proposed method with 1) a full POMDP formulation for this problem as introduced by us in [19], 2) the dual execution framework as introduced by Toussaint et. al. [1]. The full POMDP formulation is equipped with a set of macro actions, which is formulated similarly to the one in work [8], [12]. Each macro action correspond to one QMDP policy [20] (POMDP policy using Q-function approximation) that is a policy solved for one sampled particle. The POMDP synthesis policy is built as a finite state controller using one-step look-ahead. Therefore, the value function of next belief node is estimated through QMDP

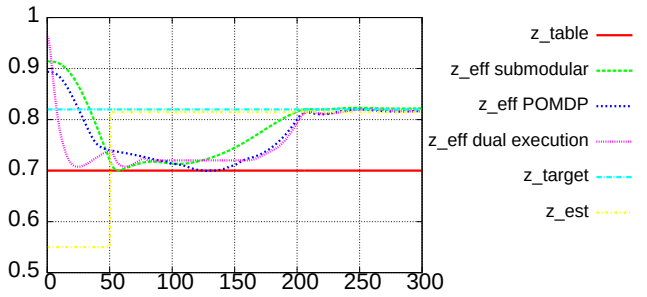


Fig. 3. Typical runs from three algorithms in hand-table. x-axis show time steps, y-axis shows the end-effector's z coordinate.

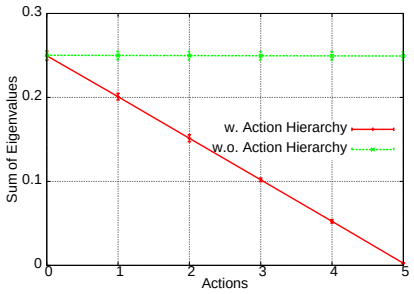


Fig. 4. Relative uncertainty reduction of two algorithms, the means and 0.95% confidence intervals are averaged over 20 runs.

function approximation [20], and information gathering is done via one-step look-ahead.

2) Peg-In-Hole-Like: In this problem, we sample 1000 particles of tuple (w, l, x, y, z) (see III-C.2 for their definitions) from a multivariate normal distribution with the ground truth as means and a covariance matrix $\Sigma = diag(0.1, 0.1, 0.1, 0.1, 0.1)$. The action hierarchy is as given in Fig. 1. The *lower-level* subtask *Detect_Z* consists of 10 different trajectories as similar as in the *hand-table* problem. The *lower-level* subtask *Detect_E* consists of 100 different trajectories. The action selection at the *higher-level* subtasks, i.e. *Root* node, is executed offline using either POMDP or adaptive submodularity. All trajectories of *Detect_E* start at the contact point with the table, which is the terminal point of the *Detect_Z* execution. Each different trajectory that is a linear motion with different movement directions is optimized to keep contacts with the table's surface using a force controller as described in [1].

We compare two adaptive submodularity methods: one with an action hierarchy (the proposed framework), and one without an action hierarchy as introduced by Javdani et. al. [10].

B. On Simulated KUKA Arm

1) Hand-Table Problem: First we provide the typical runs as drawn in Fig. 3 from three compared algorithms. The ground truth table's height is 0.7 meter, hence the goal position is at 0.82 meter height. Comparing between the adaptive submodularity and dual-execution approach, the policy computed by adaptive submodularity requires table contacts little enough to gain certainty, therefore the end effector releases the table's surface early to reach the goal that has been localized. The dual-execution method used a heuristic way to combine uncertainty reduction via reward funneling and movement cost, hence an optimal moment to release the contact was not optimized.

Comparing with the POMDP policy, the adaptive submodularity's policy is suboptimal in terms of finding the trajectory with the least cost. As our adaptive submodularity formulation is designed to first detect the table's surface, therefore path planning is not taken into account. On the other hand, POMDP is a principle framework that can optimize uncertainty reduction and path planning altogether. However, the computational time of the POMDP policy is more than 50 seconds compared to one second in total of the adaptive submodularity approach.

Figure 5(a) depicts 10 typical runs of the adaptive submodularity's policy with different ground truth table's heights.

2) Peg-In-Hole-like: Figure 6 (top row) shows a typical run of the adaptive submodularity approach with being given an action hierarchy. As shown in Section III-C in detail, the adaptive submodularity policy selects first *Detect_Z*, then *Detect_E*. This is the same as the POMDP policy computed offline at the *Root* node. The POMDP formulation given the estimate model is so simple that it can be computed by the value iteration method.

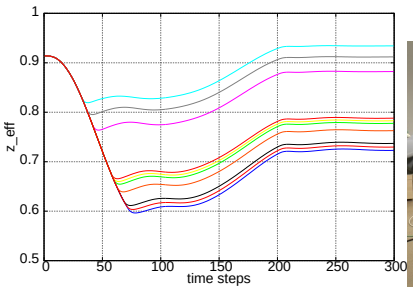
Figure 4 reports the relative uncertainty reduction of two adaptive submodularity algorithm with and without an action hierarchy. The means and 95% confidence intervals are averaged over 20 runs with different ground truth, sampled particles Φ, and sampled trajectories $\mathcal{A}$ at the *lower-level* subtasks. The adaptive submodularity method without action hierarchy could reduce uncertainty very slowly. Because, most of the time an action terminates with no contact, therefore it could prune only one hypothesis from the particle set. Therefore, in order to reduce all uncertainty it needs to execute $O\big((N+1)/2\big)$ number of actions in average as computed in Eq.16.

C. On Willow Garage PR2 Robot Platform

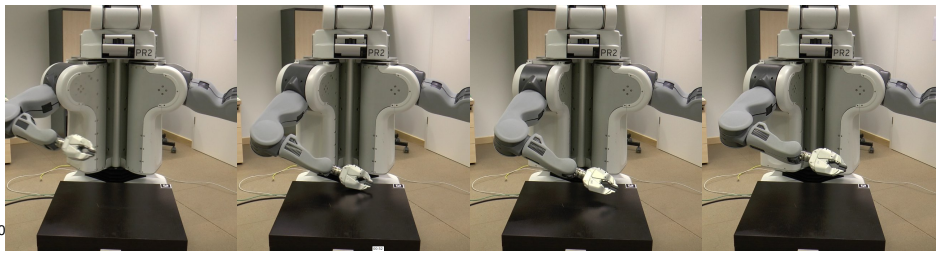
We implemented our proposed algorithm on a Willow Garage PR2 robot platform. We used only the right arm, which is 7 DoF, to solve both *hand-table* and *peg-in-hole-like* problems. The experiment setup is as shown in Fig. 5(b) and 6 (bottom row). We reported a typical run in each problem instance. The contact is sensed via a F/T sensor at the wrist. The experimental results show that the proposed algorithms are easily and successfully reimplemented on the physical system. Please see also the attached video.

V. Conclusion

The method proposed in this paper was inspired by the advantages of hierarchical POMDP planning and adaptive submodularity in tackling challenging manipulation problems under uncertainty. As adaptive submodularity can guarantee a near-optimal performance with a greedy policy, it provides a powerful framework for online sequential decision making. Being integrated with an action hierarchy, adaptive submodularity can deal with more complex problems like hierarchical POMDPs, particularly with problems that require exploration actions more rigorously. Through the integration of an action hierarchy we can exploit contact information to disambiguate uncertainty in a hierarchical way such that the problem is



(a) 10 typical runs



(b) A typical run on a physical PR2 platform: the robot end-effector starts above the table; it moves down and touches the table's surface; it starts to release the table's surface; it reaches the goal.

Fig. 5. Hand-table problem: typical runs.

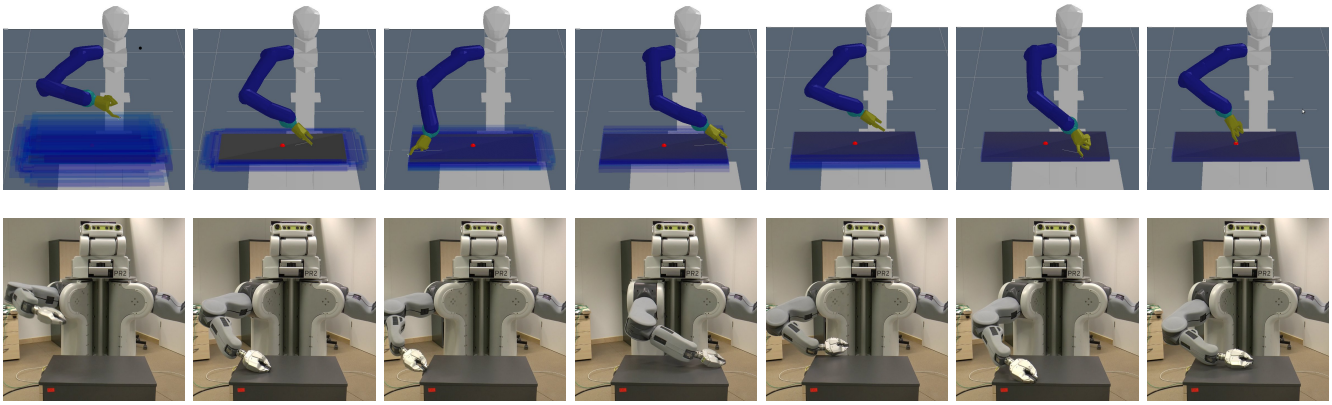


Fig. 6. Peg-in-hole-like problem (from left to right, the same explanation for both top and bottom rows): 1) uncertainty as drawn with transparency at the initial pose; 2) After executing *Detect_Z* and terminating with the first contact, the hypothesis set is pruned, the height certainty is gained; 3) continue with *Detect_E*, moving left while keeping table contacts until losing, the left edge is detected; 4) similarly, the right edge is detected; 5) the front edge is detected; 6) the rear edge is detected; 7) after the termination of *Detect_E* the table is localized, the goal is reached.

solved more effectively, as in hierarchical POMDP planning. The experimental results in both a simulated robot arm and a physical PR2 robot platform have shown the success and efficiency of the proposed framework.

REFERENCES

[1] M. Toussaint, N. Ratliff, J. Bohg, L. Righetti, P. Englert, and S. Schaal, "Dual execution of optimized contact interaction trajectories," in *IROS*, 2014.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *International Journal of Robotic Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[3] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Robust grasping under object pose uncertainty," *Autonomous Robots*, vol. 31, no. 2-3, pp. 253–268, 2011.

[4] I. Mordatch, E. Todorov, and Z. Popovic, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics*, vol. 31, no. 4, p. 43, 2012.

[5] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processses," *Mathematics of Operations Research*, vol. 12, no. 3, p. 441450, 1987.

[6] R. He, E. Brunskill, and N. Roy, "PUMA: Planning under uncertainty with macro-actions," in *AAAI*, 2010.

[7] N. A. Vien and M. Toussaint, "Hierarchical Monte-Carlo planning," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, Austin, USA*, 2015.

[8] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, "Monte Carlo value iteration for continuous-state POMDPs," in *WAFR*, 2010, pp. 175–191.

[9] D. Golovin and A. Krause, "Adaptive submodularity: Theory and applications in active learning and stochastic optimization," *Journal of Artificial Intelligence Research (JAIR)*, vol. 42, pp. 427–486, 2011.

[10] S. Javdani, M. Klingensmith, J. A. Bagnell, N. S. Pollard, and S. S. Srinivasa, "Efficient touch based localization through submodularity," in *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, 2013, pp. 1828–1835.

[11] J. Pineau, "Tractable Planning Under Uncertainty: Exploiting Structure," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, 2004.

[12] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Grasping pomdps," in *2007 IEEE International Conference on Robotics and Automation, ICRA 2007, 10-14 April 2007, Roma, Italy*, 2007, pp. 4685–4692.

[13] D. Hsu, W. S. Lee, and N. Rong, "What makes some POMDP problems easy to approximate?" in *Twenty-First Annual Conference on Neural Information Processing Systems (NIPS)*, 2007, pp. 689–696.

[14] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions," *I. Mathematical Programming*, vol. 14, no. 1, p. 265294, 1978.

[15] L. A. Wolsey, "An analysis of the greedy algorithm for the submodular set covering problem." *Combinatorica*, vol. 2, no. 4, pp. 385–393, 1982.

[16] A. Krause, A. P. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, pp. 235–284, 2008.

[17] M. Toussaint, "Newton methods for k-order Markov constrained motion problems," *CoRR*, vol. abs/1407.0414, 2014.

[18] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998.

[19] N. A. Vien and M. Toussaint, "POMDP manipulation via trajectory optimization," in *In Proc. of the Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[20] M. Hauskrecht, "Value-function approximations for partially observable markov decision processes," *J. Artif. Intell. Res. (JAIR)*, vol. 13, pp. 33–94, 2000.